半順序による証明手続きの解析

An Analysis of Theorem Proving with Partial Orders

越村三幸　瀧和男
(財) 新世代コンピュータ技術開発機構研究所

概要

証明手続きの並列化について述べる。証明手続きとしては LK [Gentzen] に基づくもの、計算モデルは半順序[Scott] によるものを考える。半順序の考えを用いて逐次アルゴリズムから段階的に並列アルゴリズムを導く。そして、導かれた並列アルゴリズムについて考察する。

1　はじめに

本論文では、LK(logisticscher Kalkül) に基づく 1 階述語論理の証明手続きの並列性について考察する。対象となる問題領域は特に固定せず、証明手続きの核になる原理的な部分のみを扱う。

並列性を考える時、始めから大きなシステム全体を考察すると本質を見失いがちである。というのはそのような大きなシステムでは、様々な最適化や機能拡張が行われているからである。このような付加的なものを除去したシステムの核となるような部分についての考察から始め、段階的に機能拡張や最適化を行うのが順当である。本論文は、並列証明機の核となる部分について考察する。

逐次計算は全順序集合上の事象を計算するのに対し、並列計算は半順序集合 (poset) 上の事象を計算する。つまり逐次計算は全順序集合上に展開され、並列計算は半順序集合上に展開されるものと見なすことができる。このような観点からの並列計算モデルが半順序によるモデル[Scott, Pratt] である。この基本的考え方は半順序関係$\leq$を事象の集合に導入することである。事象 e, f 間に $e \leq f$ の関係があるとき、e は f より先に発生すると解釈され、順序付けされていない事象は並列に発生してもよいと解釈される。本論文では半順序の観点から問題を観察し、問題解決手続きの半順序性を見つけだし並列アルゴリズムを獲得していくことを試みる。

最初に、LK に基づく形式的体系[Schütte] を与え、そしてこの体系に基づいた逐次的証明手続きを述べる。そして半順序モデルの観点から、逐次的証明アルゴリズムから段階的に並列アルゴリズムを導く。最後に、得られた並列アルゴリズムについての考察を行う。

2　形式的体系

本論文で扱う体系は、[Schütte] と同じである。Appendix で簡単な解説をする。「4 並列化」では証明図の半順序性に着目することにより並列アルゴリズムを考察していく。

3　証明手続き

3.1　基本的考え方

証明手続きの基本的考え方は、推論規則をさかさまに眺めることである。つまり、推論規則に対応した分解規則を考えることである。

$$\frac{S_1}{S} \Rightarrow \frac{S}{S_1} \qquad \frac{S_1 \quad S_2}{S} \Rightarrow \frac{S}{S_1 \quad S_2}$$

証明すべき式を受け取ったら、分解規則に従ってそれを分解する。そして、得られた式に対して、手続きを再帰的に適用する。これを、式が公理になるまで繰り返す。

こうして、得られた分解木を逆に眺めると、証明図が得られる。分解木は最初の式の恒真性を示す。つまり、式が証明されたことになる。

命題論理においてはこの方法で真偽の決定が可能であるが、述語論理においては、この手続きは終了しないことがある。与えられた式が恒真なら、この手続きは成功裏に終了する (完全性) が、恒真でない式を与えられると一般に終了しないことが知られている。

3.2 実現上の考察

3.1における問題点をあげる。

3.2.1 cut

cut 推論は用いない。これは cut-formula を推定するのが困難だからである。しかし基本定理により、証明できる範囲が狭くなるようなことはない。

定理 1 (基本定理) [Gentzen, Schütte] 式Sが cut を用いて証明可能なら、Sは cut を用いないで証明可能である。

3.2.2 ∃L と ∀R

∃L と ∀R の quantifier を分解する時は、束縛変数を新しい (下式に現れない) 自由変数で置き換える。ここに大きな問題はない。

3.2.3 ∀L と ∃R

∀L と ∃R は t の値として何を選択するかが問題となる。t の値を選択する方法として以下の方法が考えられる[Edwald]。

(a) Herbrand universe による方法: 下式の Herbrand universe から λ 個の要素を選ぶ。非現実的な方法である。

$$\frac{\forall x A(x), A(t_1), A(t_2), \ldots, A(t_\lambda), \Gamma \rightarrow \Theta}{\forall x A(x), \Gamma \rightarrow \Theta}$$

t_iは下式の Herbrand universe の要素

(b) 導出原理による方法: 導出を行い t の値を決める[Miller]。

(c) t 変数による方法: 取り敢えず値を決めず、仮の変数 (t 変数) を代入しておきあとで値を決める。

本論文では、(c) について考察する。

∀L と ∃R の時、新しい (下式に現れない) t 変数で束縛変数を置き換える。

$$\forall L: \frac{\forall x A(x), A(t), \Gamma \rightarrow \Theta}{\forall x A(x), \Gamma \rightarrow \Theta} \qquad \exists R: \frac{\Gamma \rightarrow \Theta, A(t), \exists x A(x)}{\Gamma \rightarrow \Theta, \exists x A(x)}$$

t は t 変数で下式に現れない。t 変数は、「3.2.4 大域的ユニフィケーション」で役割が明らかになる。t 変数の扱いには、若干の注意が必要である[Edwald]。

3.2.4 大域的ユニフィケーション

一般に、t 変数の値は次のように局所的には決定できない[1]。

$$\dfrac{\dfrac{B(t), A(a) \rightarrow B(b) \quad A(a) \rightarrow B(b), A(t)}{A(t) \supset B(t), A(a) \rightarrow B(b)} \ (\supset L)}{\forall x(A(x) \supset B(x)), A(a) \rightarrow B(b)} \ (\forall L)$$

左の上式は置換 b/t(tをbで置き換える) で公理になり、右の上式は置換 a/tで公理となる。これは同時には起こり得ない。このようなチェックは局所的には (一つの式だけに着目しているだけでは) 行えない。したがって、t 変数の値の決定は大域的に行われなければならない。この操作を大域的ユニフィケーションと呼ぶことにする。

大域的ユニフィケーションは次の 2 つの段階からなる。

[1] Appendix の $\forall L$ の推論規則では上式に quantifier が残っていたがここでは議論に関係ないので省略した。

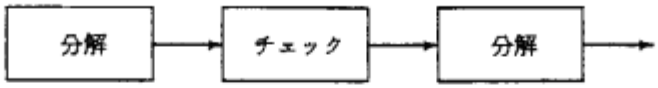

図 1: 分解段階とチェック段階

1. (局所的) 置換生成: 各式 S_i を公理にするような t 変数の置換の集合 $\Theta_i = \{\theta_i^1, \theta_i^2, \ldots, \theta_i^{n_i}\}$ を求める。ここで、$\forall j(1 \leq j \leq n_i)S_i\theta_i^j$ は公理、が成り立つ。

2. (大域的) チェック: $\{\Theta_i | 1 \leq i \leq n\}$ から無矛盾な n 個の置換の合成の集合 $\Pi_{i=1}^{n}\Theta_i = \{\Theta = \Pi_{i=1}^{n}\theta_i^{j_i} | \forall i(1 \leq i \leq n)S_i\Theta$ は公理$\}$ を生成する。

ここで二つの置換 θ, λ の合成を $\theta \circ \lambda$ で表し [Chang]、$\Pi_{i=1}^{n}\theta_i^{j_i} = \theta_1^{j_1} \circ \theta_2^{j_2} \circ \ldots \circ \theta_n^{j_n}$ とする。この操作は、整合ラベリング問題[Haralick] と同じである。

3.3 証明アルゴリズム

以上述べたことより証明アルゴリズムは、分解段階 (局所的置換生成を含む) と大域的チェック段階からなる。

decompose(式) 分解段階は証明木を構成する段階であり概略次のようになる。

(1) 式が公理の時、**return**(true)

(2) [2] 式を分解するような分解規則があり、その規則の上式が 式 1(, 式 2) の時、
if decompose(式 1) = fail **then return**(fail) **else return**(decompose(式 2))

(3) 式を公理にするような (t 変数の置換) $\theta_1, \theta_2, \ldots, \theta_n$ がある時、式と $\theta_1, \theta_2, \ldots, \theta_n$ をデータベースに登録する。**return**(true)

(4) (1),(2),(3) のいずれでもない時、**return**(fail)

gcheck 大域的チェックは分解段階で得られたデータベース中の Θ_i より $\Pi_{i=1}^{n}\Theta_i$ を求めるものである。

(1) データベースが空の時、**return**(true)

(2) $\Pi_{i=1}^{n}\Theta_i$ が空でない時、**return**(true)

(3) $\Pi_{i=1}^{n}\Theta_i$ が空の時、**return**(fail)

prove(式) トップレベルは次のようになる。

(1) **if** decompose(式) = fail **then return**(fail)

(2) **if** gcheck = true **then return**(true)

(3) (1)(2) 以外の時、データベースに式 1, 式 2,..., 式 n の n 個の式が登録してあるものとする。
for i = 1 **to** n **if** decompose(式 i) = fail **then return**(fail);
(2) に行く。

手続きの流れを図示すると次のようになる (図 1)。分解段階とチェック段階が交互に現れていることが分かる。

この手続きは、恒真の式に対しては証明に成功して停止するが、恒真でない式に対しては、停止するとは限らない。したがって、(2)(3) の繰り返しの回数を工学的に制限する必要がある。こうすると手続きの完全性は失われる。回数を n 回に制限した時に証明不可能だが、$n+1$ 回で証明可能な式を構成することができる。

[2] 推論規則 $\forall L, \exists R$ は上式と下式の論理記号の数が変化しない。したがって、制御が (3) に移らないことがあり得るので実現上注意が必要である。

4 並列化

Edwald は、「3 証明手続き」で述べた方法の素直な並列化について簡単に考察している[Edwald]。ここでは事象間の半順序に着目し、より詳細に並列性を考察する。

4.1 並列計算 − 半順序

計算中の事象はその因果関係において半順序を成すことが多い。しかし逐次計算では、その事象に全順序を与えなければならなかった。そして、アルゴリズムを考える時には全順序性を考慮に入れなければならなかった。そのことが、アルゴリズムを考える上での大きな制約になっていた。

一方並列計算では事象を半順序で実行することが許される。したがって、事象生来の因果関係を反映させてより自然に実行することが可能となる。このことは、アルゴリズムを考える上での全順序という制約が取り除かれたことを意味し、新しいアルゴリズムの発見も期待される。半順序によって並列計算が自然にモデル化でき[Pratt]、また半順序の考えを取り入れた計算モデルを持つ言語[Yoshida] も提案されている。

計算を構成する事象の集合に半順序関係 $\leq$ を導入し、事象 e, f 間に $e \leq f$ が成り立つ時、e は f より先に起こると解釈する。そして、順序付けされない事象は並列に起こっても良いと解釈する[Pratt]。任意の事象より$\leq$ の意味で最小の事象を仮想的に考えこれを⊥で表す。最大の場合も同様に事象⊤を考える。事象⊥は計算の開始、事象⊤は計算の終了を表し、⊥の状態が計算の初期状態、⊤の状態が計算の終了状態つまり計算結果を表すものと見なすことができる。

逐次計算は半順序を適当に全順序化したものと見なすことができる。また、ハードの制約や探索問題の枝刈り等による部分的な全順序化も同様である。

本論文では、状態は事象間の関係と見なす。非形式的にいうと、非循環有向グラフのノードが事象, アークが状態に対応する。

事象 事象を、頂点の集合$\mathcal{V}$, アルファベットの集合Σ, $\mathcal{V}$上の半順序$\leq$, ラベル付け関数 $\mu : \mathcal{V} \mapsto \Sigma$ の四つ組 $< \mathcal{V}, \Sigma, \leq, \mu >$ で定義する。μ は単射である必要はない。$\mathcal{V}$ は事象をモデル化したもの、Σ は動作を表している。$v_1, v_2 \in \mathcal{V}$ に対して $v_1 \leq v_2$ は事象 v_1 は事象 v_2 より先に発生する必要があると解釈する[Pratt]。

次にプロセスの定義をするために、連結な半順序集合を定義する。集合 X 上の半順序を $\leq$ とし、$x \approx y \stackrel{def}{\Longleftrightarrow} x \leq y \vee y \leq x$ とする。$x, y \in X$ に対して系列 $x = x_0 \approx x_1 \approx \ldots \approx x_n = y \; for \; \forall i (0 \leq i \leq n) x_i \in X$ が存在する時、x と y は連結であるという。X の任意の二つの元が連結の時、X を連結という。

プロセス プロセスは$< \mathcal{V}', \Sigma, \leq, \mu >$ の形で表される。ここで、$\mathcal{V}'$ は $\leq$ に関して連結な $\mathcal{V}$ の部分集合である。

次に状態の定義をするために、素商と組成列の定義をする。集合 X 上の半順序を $\leq$ とする。$x, y \in X$ に対して $x > y$ が成立し $x > z > y$ となるような $z \in X$ が存在しない時, x と y の対を素商と呼び、$x/_p y$ と表記する。また、X の元の列 $x_0, x_1, \ldots, x_k$ で $x_{i-1}/_p x_i (1 \leq i \leq k)$ が成り立つ時、$x_0, x_1, \ldots, x_k$ を組成列という。

$\mathcal{V}$ の ($\leq$ に関する) 素商の集合を $\mathcal{S}$ とし、$\mathcal{S}$ 上の 2 項関係 $\preceq$ を $(v_1^1/_p v_1^0) = s_1 \preceq s_2 = (v_2^1/_p v_2^0)(s_i \in \mathcal{S}, v_i^j \in \mathcal{V})$ $\stackrel{def}{\Longleftrightarrow}$ $\mathcal{V}$ の ($\leq$ に関する) 組成列 $v_2^1, v_2^0, \ldots, v_1^1, v_1^0$ が存在する、と定義する。$\mathcal{S}$ は $\preceq$ に関して半順序となることが容易に分かる。

状態 $\mathcal{S}$, アルファベットの集合Σ', $\mathcal{S}$上の半順序$\preceq$, ラベル付け関数 $\mu' : \mathcal{S} \mapsto \Sigma'$ の四つ組 $< \mathcal{S}, \Sigma', \preceq, \mu' >$ で状態を定義する。μ' は単射である必要はない。$\mathcal{S}$ は状態をモデル化したもの、$s_1, s_2 \in \mathcal{S}$ に対して $s_1 \preceq s_2$ は状態 s_1 は状態 s_2 より先になる必要があると解釈する。計算の状態は $\preceq$ に関して比較不能な元から成る $\mathcal{S}$ の部分集合である。

例 1 (クイック・ソート) クイック・ソートの半順序性を考える。ソートはオブジェクトの列を受け取りそれをある全順序に従って並べ替えるものである。クイック・ソートの Prolog プログラムは次のようになる。append/3,partition/4 は組み込みの機能であると仮定する。

```
qsort(□,      □).
qsort([X|Xs], Ys) :- partition(X,Xs,Ss,Ls),
                     qsort(Ss,Ys1),
                     qsort(Ls,Ys2),
                     append(Ys1,[X|Ys2], Ys).
```

qsort/2 はオブジェクトの列を第 1 引数に受け取り、ソートした結果を第 2 引数に返す。partition/4 は第 2 引数のオブジェクトの列を第 1 引数のオブジェクトより小さいオブジェクトの列 (第 3 引数) と大きいオブジェクトの列 (第 4 引数) に分割するものである。

[2,3,1]をソートした時の事象の関係は図 2 のようになる。ノードは事象 (qsort, append 等でレベル付けされている)、アークは状態 ([2,3,1],2 等でレベル付けされている) を表す。例えば一番左側の状態の[3,1] は事象 qsort と事象 part の関係、一番左側の事象 qsort は状態間の対応([2,3,1]) qsort (2,[3,1],2) と見なせる。また点線で表した計算状態は(1,1,□,□,□,3,2)である。以後、解説に用いる図のノードは事象、アークは状態であるとする。

事象の poset の長さ (length) は計算時間に幅 (width) は並列度に相似なものと見なすことができる。例えば図 2 では長さは 6 、幅は 4 である。実際の並列計算ではプロセッサ数やプロセッサ間通信オーバヘッドの制約の下、できるだけ幅を大きくし長さを短くすることが処理効率上要求される。

以上のように並列の利点は、事象の自然な実行と処理速度の高速化の 2 点にあるものと思われる。

4.2 並列アルゴリズム 1

「3.3 証明アルゴリズム」で述べたように証明過程は、証明図を構成する分解段階と大域的チェック段階からなる。証明図中の式の集合が $\preceq$(Appendix 参照) に関して poset であることから、自然に以下のような並列アルゴリズムが得られる。分解段階には分解プロセス、大域的チェック段階には大域的チェックプロセスが対応し、分解プロセスのアルゴリズムは次のようになる。

pdecompose1(式)

(1) 式が公理の時、終了。

(2) 式を分解するような分解規則があり、その規則の上式が 式1(, 式2) の時、 pdecompose1(式1) (||pdecompose1(式2))

(3) 式を公理にするような置換 $\theta_1, \theta_2, \ldots, \theta_n$ がある時、 $\{\theta_1, \theta_2, \ldots, \theta_n\}$ を大域的チェック・プロセスに送り、終了か継続かの結果を待つ。

  (a) 結果が終了の時、終了。

  (b) 結果が継続の時、(2) に行く。

(4) (1),(2),(3) のいずれでもない時、fail を大域的チェック・プロセスに送り終了。

|| は並列実行を表す[Pratt]。

pgcheck 大域的チェック・プロセスは次のようになる。

(1) fail が通知されたら、各分解プロセスに終了を通知し終了する。

(2) $\Pi_{i=1}^{n}\Theta_i$ が空のとき、各分解プロセスに継続を通知する。

(3) $\Pi_{i=1}^{n}\Theta_i$ が空でない時、各分解プロセスに終了を通知し終了する。

pprove1(式) pdecompose(式) || pgcheck

これは、[Edwald] と一致する (図3)。分解段階が全順序から半順序になったのが分かる。

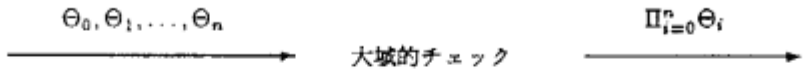

図 4: 大域的チェック

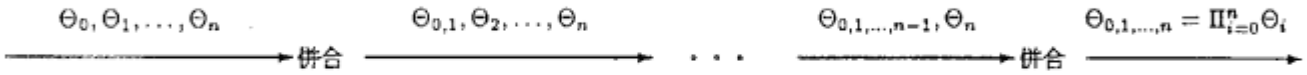

図 5: 併合

4.3 並列アルゴリズム 2

「4.2 並列アルゴリズム 1」で述べた方法ですぐ気がつくのは、大域的チェック段階が全順序つまり逐次であるのが分かる。この部分は NP 完全なので、この部分に対してもう少し詳しく検討していくことにする。

大域的チェックは置換の集合$\Theta_i = \{\theta_i^1, \theta_i^2, \ldots, \theta_i^{n_i}\}$ の集合 $\{\Theta_i | 1 \leq i \leq n\}$ から無矛盾な置換の組み合わせ $\Pi_{i=1}^{n}\Theta_i$を探索するものであった (図 4)。これは二つの置換の集合を一つの置換の集合にまとめる併合操作を繰り返し、最終的に一つの置換の集合を得ることでできる。この方法は整合ラベリング問題では併合法と呼ばれている解法である[西原]。

この併合操作は具体的には二つの置換の集合 Θ_1, Θ_2 から無矛盾な置換の集合 $\Theta_{1,2} = \{\theta_1^{m_1} \circ \theta_2^{n_2} | \theta_1^{m_1} \in \Theta_1, \theta_2^{n_2} \in \Theta_2\}$ を生成する操作である。そして図 5 のような系列が得られる。系列の順序に特に意味はない。つまりこの部分は全順序のままである。

ここで再び証明図の半順序性を利用する。証明図の poset と双対の poset を考えこれの極小でない元に併合操作を行う事象を割り当てる (図 6)。こうすることによって分解段階とほぼ等しい並列度が期待される。

pdecompose2(式)

(1) 式が公理の時、終了。

(2) 式を分解するような分解規則があり、その規則の上式が 式 1(, 式 2) の時、 pdecompose2(式 1) (|| pdecompose2(式 2) || 併合プロセス)

(3) 式を公理にするような置換 $\theta_1, \theta_2, \ldots, \theta_n$ がある時、 $\{\theta_1, \theta_2, \ldots, \theta_n\}$ を直近の併合プ ロセスに送り、終了か継続かの結果を待つ。

  (a) 結果が終了の時、終了。

  (b) 結果が継続の時、 (2) に行く。

(4) (1),(2),(3) のいずれでもない時、 fail を直近の併合プロセ スに送り終了。

併合プロセス

(1) fail を通知されたら、(自分の受け持ちの) 各分解プロセスに終了 を通知し直近の併合プロセスに fail を通知し終了。

(2) $\Theta_{i,j}$ が空の時、 (自分の受け持ちの) 各分解プロセスに継続を通知し、直近の併合プロ セスに空集合を報告し終了する。

(3) $\Theta_{i,j}$ が空でない時、 直近の併合プロセスにΘ_i, Θ_jを報告し終了する。

最終の併合プロセスは若干異なった操作を必要とする。

最終併合プロセス

(1) fail を通知されたら、(自分の受け持ちの) 各分解プロセスに終了 を通知し終了。

(2) $\Theta_{i,j}$ が空の時、(自分の受け持ちのつまり全ての) 分解プロセスに継続を通知し、実行を継続する。

(3) $\Theta_{i,j}$ が空でない時、(自分の受け持ちのつまり全ての) 分解プロセスに終了を通知し、終了する。

5 考察

5.1 自然性

証明図の半順序性に着目することによって、分解段階と大域的チェック段階にわたる並列化が自然にできた。自然ということは、問題が生来もつ並列性を引き出したことを意味するものと思われる。

5.2 証明図の構成に対応した並列性

西原らは、一般的な概整合ラベリング問題の併合法の効率の詳細な検討と評価を行っている。併合操作の効率は併合操作を施す順序 (併合系列) にかなり依存する。併合系列とは集合 $\{\Theta_i | 1 \leq i \leq n\}$ の要素に全順序を与えたもので、その順序で併合が進められる (図 5)。効率の良い併合系列を求める方法が研究されている [西原]。効率の尺度は計算途中で生じる中間解の個数 (例えば$\Theta_{1,2}$の元の個数) である。

「4.3 並列アルゴリズム 2」で述べた大域的チェック段階での併合操作の並列化を、松尾らは (併合法の) 構造レベルの並列化と呼び、その有効性についてメモリ共有型並列計算機での実験を報告している[松尾]。これは一般的な概整合ラベリング問題を扱っており、具体的な問題ごと (本論文の場合は証明手続き) に固有の構造に基づいた並列化ではなく、 syntactical な構造に基づいた並列化について考察している。これによると、構造レベルの並列化は場合によっては非効率的であることが確かめられている。その理由は中間解の個数が増大する場合があるからである。

この効率の問題は、半順序の併合系列の与え方に依存している。松尾らはこの半順序の与え方を特に指定していない。並列計算を考える場合は半順序の併合系列についての考察も必要になると思われる。並列アルゴリズム 2 で述べた方法は、証明図という問題固有の構造に基づいた並列化を行っており、半順序付けは自然であり、半順序付けのためのオーバヘッドもない。

Θ_i と Θ_j を併合する時に、 Θ_i と Θ_j に含まれる共通の t 変数の割合が大きいほど、中間解 ($\Theta_{i,j}$の元) の個数は平均的に少なくなる。本方法で$\Theta_{i,j}$の操作が行われるのは対応する証明図の '近い' もの順である。 t 変数の生成機構と最終的な解は通常一つか数個であることを考えれば、併合操作の初期の段階に解 (置換) が絞りこまれていくことが、つまり中間解はそれほど増大しないことが予想される。

5.3 AND 並列と OR 並列

「3.3 証明アルゴリズム」の decompose 手続き (2) における選択点は二つある。一つは分解した後どちらの上式の分解を先に進めるかという点、もう一つはどの分解規則を適用するかという点である。前者は AND 並列、後者は OR 並列に対応する。

本論文では前者を扱ってきた。後者は証明戦術に深く関係する。どの分解規則を適用するかは、通常ヒューリスティクに決まり、万能な戦術は今のところ知られていない。このところを補完するのに、OR 並列は有効であると思われる。また、恒真でないあるクラスに属する式に対しての停止性をも扱う証明機を構成する場合には、 AND 並列も重要な戦術の対象になる。 AND-OR 並列を踏まえた戦術の記述法は、今後の課題である。

5.4 導出原理との関係

t 変数法による証明法を導出原理 [Robinson, Chang] による証明法に翻訳すると次のようになる。導出原理では節の集合から導出を行い空節を導くことで証明を行う。この最初の節の集合をC_0とする。これをもとに節の集合の系列 $C_0, C_1, \ldots, C_n, C_{n+1}, \ldots$ を生成する。各 C_n は次の条件を満たす： $C_n = \{c | c$ は節の集合 $\{c_1, c_2, \ldots, c_m | c_i \in C_0\}$

から導出[3]されるが c_i の重複は高々 n 回である}。空節を含むようなC_kが得られた時点で、証明に成功して終了する。

導出原理におけるこの証明法は、公平でかつ地道の感じさえする。しかし、導出原理においても万能な戦術は報告されていないことを考えれば、このような方法を基礎にした戦術を考察するのも興味深い。

また導出原理では並列化のしにくいと思われるのに、LK では自然に並列化できたのは式という分散データベースがあり、それらが証明図を反映して構造化されているためであろう。

6 おわりに

問題に内在する半順序性を考察することで、段階的に並列アルゴリズムを導き出した。自然に導き出せたのは定理証明というよく形式化された問題を扱ったことが大きいと思われる。他の問題の並列性を考える時、ここで述べたような方法で自然に並列性が導き出せるとは限らない。しかし、半順序性を考察することは並列計算にとって重要な要因であることには変わりない。

並列計算の利点は、アルゴリズムの自然な記述 (事象の自然な実行) と処理速度の高速化の 2 点であると述べたが、本論文では前者のみについて考察した。実際に並列計算機で実行する時には、負荷分散や粒度問題など考察しなければならない点は多い。

本論文では定理証明の核となる原理的な部分のみを考察した。原理的に可解だが実際の処理では計算量が爆発的になる問題は珍しくない。したがって、実用的な証明機を構成するためにはさらに、問題領域に関する知識の扱いや型の導入など考察する必要がある。また、問題を自然に表現するために等号の導入や多値論理[Takahashi 67b] や高階論理[Takahashi 67a] への拡張も試みていきたい。

謝辞

並列計算モデルに関して貴重な助言をいただいた吉田かおる女史に感謝します。また、草稿を読んでいただきコメントをいただいた坂井公、上田和紀、横田一正、市吉伸行の各氏に感謝します。最後に、日頃から御討論いただいている ICOT KL1 プログラミング TG のメンバーの各氏に感謝します。

参考文献

[Gentzen] G. Gentzen. *Untersuchungen über das logische Schließen I,II*, Matematische Zeitschrift, 1934, Vol.39, pp.176–210,405–431

[Schütte] K. Schütte. *Beweistheorie*, Springer–Verlag, 1960

[Takahashi 67a] Moto-o Takahashi. *A proof of cut-elimination theorem in simple type-theory*, Math. Soc. Japan, 1967, Vol.19, No.4, pp.399–410

[Takahashi 67b] Moto-o Takahashi. *Many-valued logics of extended Gentzen style I*, Science Reports of the Tokyo Kyoiku Daigaku, Section A, 1967, Vol.9, No.231, pp.95–116

[Robinson] J. A. Robinson. *A Machine Oriented Logic Based on The Resolution Principle* J.ACM 12, No.1, pp.23–41

[Chang] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving* Academic Press, 1973

[Miller] Dale Miller and Amy Felty. *An Integration of Resolution and Natural Deduction Theorem Proving*, AAAI-86, 1986, pp.198–202

[Edwald] Tryggvi Edwald. *tepv- an Automatic Theorem Prover*, 筑波大学数学研究科修士論文, 1987

[3]導出に$\{c_1, c_2, \ldots, c_m | c_i \in C_0\}$の節を一回づつ使う。

[Haralick] R. M. Haralick and L. G. Shapiro. The Consistent Labeling Problem: Part I, IEEE-Trans. on Pattern Analysis and Machine Intelligence, 1979, PAMI-1, No.2, pp.173–184

[西原] 西原清一, 松尾嘉和, 池田克夫 「概整合ラベリング問題における併合法の最適化と効率評価」 人工知能学会誌, 1988, Vol.3, No.2 pp.196–205

[松尾] 松尾嘉和, 西原清一, 池田克夫 「概整合ラベリング問題における併合解法の並列化について」 人工知能学会全国大会 (第 2 回) 論文集, 1988, 1-4, pp.31-34

[Scott] Dana S. Scott. *Continuous Lattices* LNM 274, Springer-Verlag, 1972

[Pratt] Vaughan Pratt. *Modeling Concurrency with Partial Orders* International Journal of Parallel Programming, 1986, Vol.15, No.1, pp.33–71

[Yoshida] Kaoru Yoshida and Takashi Chikayama. *A'UM -A Stream-Based Concurrent Object-Oriented Language-* ICOT TR-388, 1988

Appendix

Gentzen は、syntactical な方法で今日 Getnzen の基本定理と呼ぶところのものを証明した[Gentzen]。この時使われた体系が LK である。一方 Schütte は、LK と等価な体系 (SLK) を提案しこれを用いて、基本定理の証明を semantical な方法で行った[Schütte]。ここで述べる形式的体系は、SLK を基にした体系である。

A.1 記号

体系を記述するため以下の記号を用意する。

対象定数 $c_i(i = 0, 1, 2, \ldots)$

自由変数 $a_i(i = 0, 1, 2, \ldots)$ $a, b, c, \ldots$を用いることもある。

束縛変数 $x_i(i = 0, 1, 2, \ldots)$ $x, y, z, \ldots$を用いることもある。

関数記号 $f_i^n(n = 0, 1, 2, \ldots; i = 0, 1, 2, \ldots)$ n変数関数記号

述語記号 $P_i^n(n = 0, 1, 2, \ldots; i = 0, 1, 2, \ldots)$ n変数述語記号

論理記号 $\neg, \wedge, \vee, \supset, \forall, \exists$

A.2 表現

項, 論理式及び式の定義は通常通り[Gentzen]。

項 $t_i(i = 0, 1, 2, \ldots)$

論理式 $F_i(i = 0, 1, 2, \ldots)$ $A, B, C, \ldots$を用いることもある。

式 $A_1, A_2, \ldots, A_m \rightarrow B_1, B_2, \ldots, B_n(m, n \geq 0)$ の図形を式という。内容的には、$(A_1 \wedge A_2 \wedge \ldots \wedge A_m) \supset (B_1 \vee B_2 \vee \ldots \vee B_n)$を表す。$\Gamma \rightarrow \Delta$, $S_i(i = 0, 1, 2, \ldots)$を用いることもある。 Γ, Δは論理式の列を表す (空列を含む)。論理式の集合を他にΘ, Σ等で表す。

A.3 公理

$\Gamma \cap \Delta \neq \phi$を満たす式$\Gamma \rightarrow \Delta$を公理と呼ぶ。

A.4 推論規則

$$\frac{\mathrm{S}_1}{\mathrm{S}} \quad or \quad \frac{\mathrm{S}_1 \quad \mathrm{S}_2}{\mathrm{S}}$$

の形の図を推論規則という$(n \geq 1)$。 $\mathrm{S}_1 \quad \mathrm{S}_2$を上式、 Sを下式という。

A.4.1 cut

$$\frac{\Gamma \to \Theta \qquad \Delta \to \Sigma}{\Gamma, \Delta' \to \Theta', \Sigma}$$

但し、$\Theta \cap \Delta = \{D\}$で、 $\Delta' = \Delta \setminus \{D\}, \Theta' = \Theta \setminus \{D\}$ であり Dを cut-formula という。

A.4.2 命題論理記号に関する推論

$$\wedge L : \frac{A, B, \Gamma \to \Theta}{A \wedge B, \Gamma \to \Theta} \qquad \wedge R : \frac{\Gamma \to \Theta, A \qquad \Gamma \to \Theta, B}{\Gamma \to \Theta, A \wedge B}$$

$$\vee L : \frac{A, \Gamma \to \Theta \qquad B, \Gamma \to \Theta}{A \vee B, \Gamma \to \Theta} \qquad \vee R : \frac{\Gamma \to \Theta, A, B}{\Gamma \to \Theta, A \vee B}$$

$$\supset L : \frac{\Gamma \to \Theta, A \qquad B, \Gamma \to \Theta}{A \supset B, \Gamma \to \Theta} \qquad \supset R : \frac{A, \Gamma \to \Theta, B}{\Gamma \to \Theta, A \supset B}$$

$$\neg L : \frac{\Gamma \to \Theta, A}{\neg A, \Gamma \to \Theta} \qquad \neg R : \frac{A, \Gamma \to \Theta}{\Gamma \to \Theta, \neg A}$$

A.4.3 ∀, ∃に関する推論

$$\forall L : \frac{\forall x A(x), A(t), \Gamma \to \Theta}{\forall x A(x), \Gamma \to \Theta} \qquad \forall R : \frac{\Gamma \to \Theta, A(a)}{\Gamma \to \Theta, \forall x A(x)}$$

$$\exists L : \frac{A(a), \Gamma \to \Theta}{\exists x A(x), \Gamma \to \Theta} \qquad \exists R : \frac{\Gamma \to \Theta, A(t), \exists x A(x)}{\Gamma \to \Theta, \exists x A(x)}$$

tは任意の項、aは eigen variable で下式に現れない。

A.5 証明可能

$\Gamma \to \Theta$が終式であるような証明図が存在する時、 $\Gamma \to \Theta$は証明可能であるという。以下の条件を満たす図形を証明図という。

1. この図の終式以外の式はある一つの推論規則の上式になっている。
2. 一つの式から、その下式に進んでいって元の式に戻ることはない。
3. 下式にならないような式は公理に限る。

ある推論規則の下式が S で上式が S_iのとき、 $\mathrm{S} \prec \mathrm{S}_i$と定義し、 $\mathrm{S} \preceq \mathrm{S}' \overset{\text{def}}{\iff} \mathrm{S} \prec \mathrm{S}' \vee \mathrm{S} = \mathrm{S}'$ とすると、証明図に現れる式の集合は $\preceq$ に関して半順序を成すことが分かる。

例 2 ($\to \neg(A \wedge B) \supset (\neg A \vee \neg B)$の証明)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A, B \to A \qquad A, B \to B}{A, B \to A \wedge B}\,(\wedge R)}{\neg(A \wedge B) \to \neg A, \neg B}\,(\neg L \quad \neg R)}{\neg(A \wedge B) \to \neg A \vee \neg B}\,(\vee R)}{\to \neg(A \wedge B) \supset (\neg A \vee \neg B)}\,(\supset R)$$

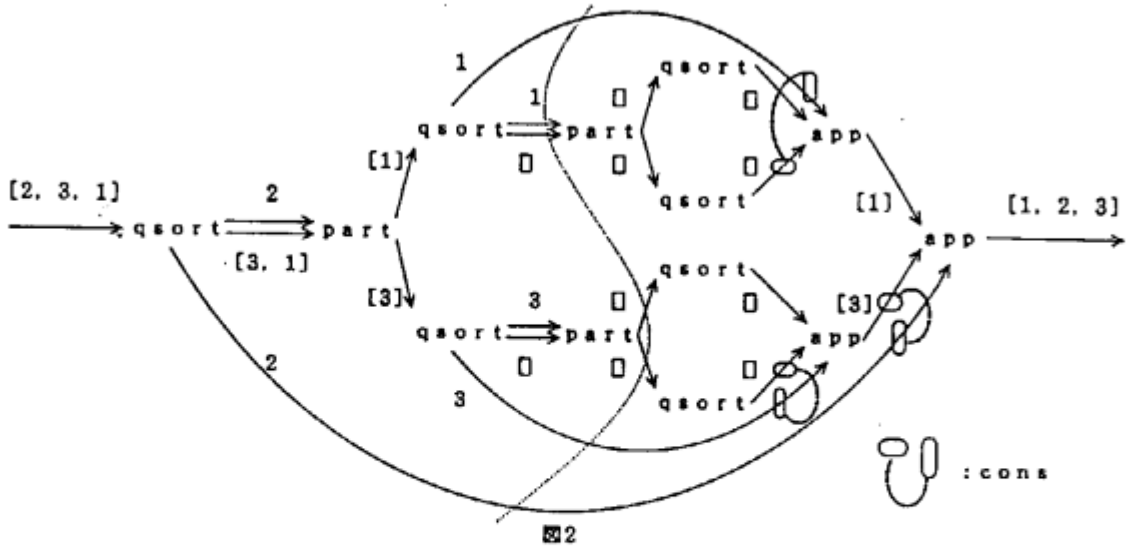
[2, 3, 1]
qsort
part
[3, 1]
[1]
[3]
1
2
3
app
[1, 2, 3]
: cons

図2

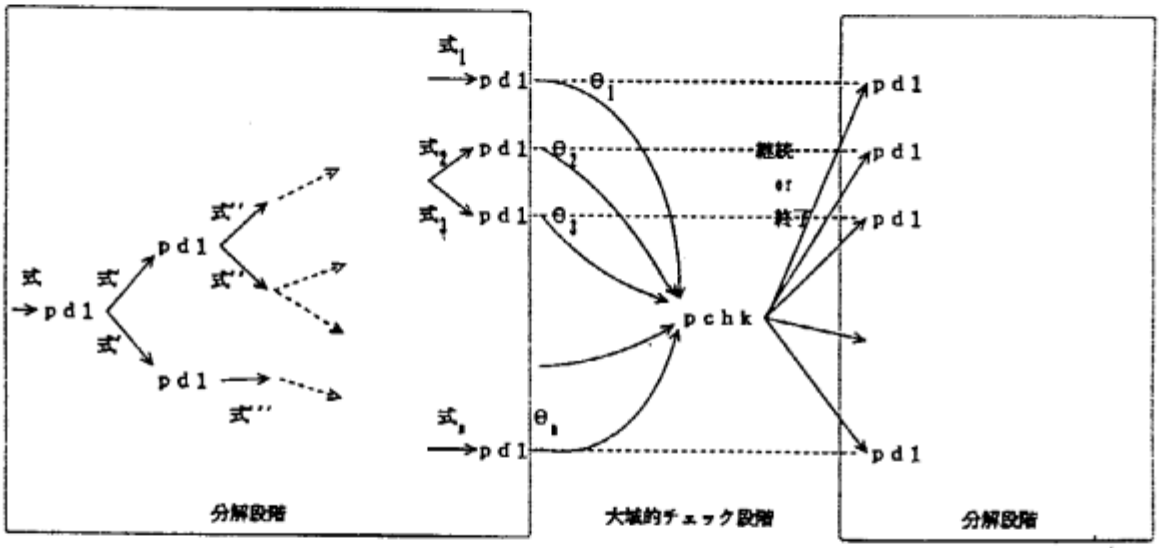
式
pd1
式'
式''
pchk
θ
継続
or
終了
分解段階
大域的チェック段階
分解段階

図3

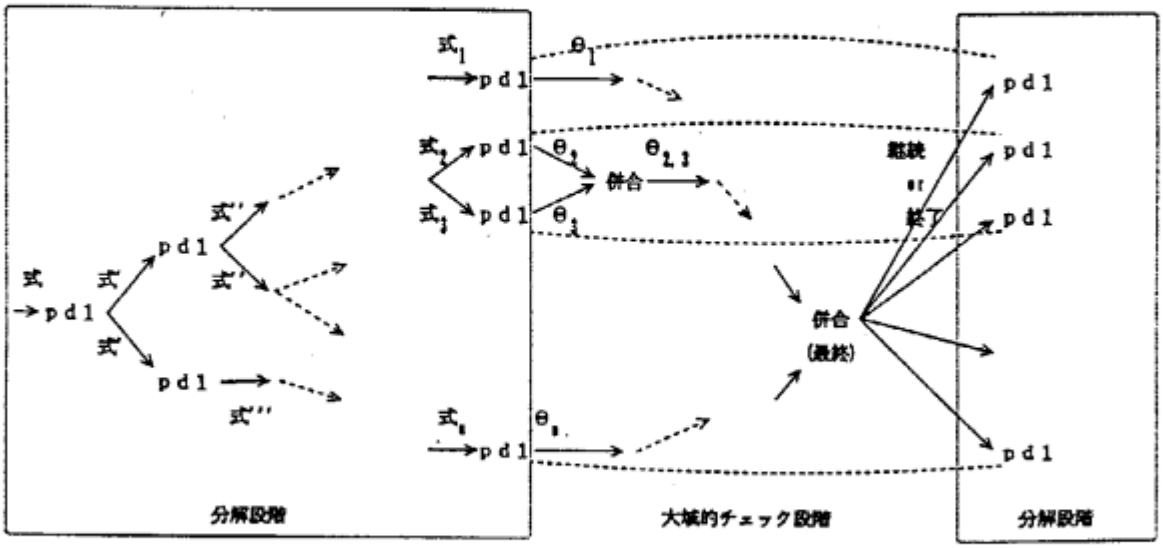
式
pd1
式'
式''
θ
併合
継続
or
終了
併合
(最終)
分解段階
大域的チェック段階
分解段階

図6